Diagnostic Data Streams

Typical assortment

Thu, Apr 6, 2006

As an aid in monitoring the activities of a front ends, a number of data streams (a term used for a formalized type of circular buffer) are installed. This note describes the functionality of many of those in common use. More extensive coverage can be found in many other notes. One early note is *Data Streams Implementation*. A more recent note is *System Enhancement*.

Installed data streams are characterized by entries in the `DSTRM` system table. The entries chosen are not significant, but they are often selected to be at the same index across most nodes, just to simplify maintenance. Here is a list of those covered herein as captured from a common IRM test node.

Index	*Name*	*DS_size*	*68K_addr*	*PPC_addr*	*Record_size*	*User_size*
0	NETFRAME	1000	109000	F04000	16	8
1	SETLOG	0800	003800	E03800	16	8
2	ACNETERR	1000	00C000	E0C000	16	8
3	TFTPLOG	0800	005800	E05800	16	8
4	EVTLOG	1000	00A000	E0A000	4	8
5	TASKLOG	1000	007000	E07000	16	8
6	AERSLOG	1000	008000	E08000	16	8
7	RETDLOG	1000	009000	E09000	16	8
8	IPNODLOG	1000	00D000	E0D000	16	40
9	ITTLOG	1000	1F2000	EF2000	16	8
10	ALLOCLOG	1000	—	EF4000	16	8
11	BUSERROR	1000	—	E0E000	16	8
12	FTPMLOG	1000	1F3000	EF3000	16	8
13	RESENDDL	1000	00E000	EF6000	16	8
19	SETDLOG	1000	1F7000	EF7000	16	8

The *Address* and the *DS_size* figures are shown in hexadecimal, so that `1000` is 4K bytes. Two data streams exist only in PowerPC nodes. During system initialization, following a front end boot, a data stream queue for each data stream described in `DSTRM` is created and initialized. Here is the `DSTRM` table entry format:

Field	*Size*	*Meaning*
`dsType`	2	Header type, always 1. When sign bit set, queue is initialized.
`eSize`	2	Record size.
`uSize`	2	User header size.
`dsSize`	2	Total size of data stream queue, including the header.
`dsAddr`	4	Base address of data stream queue.
`spare`	4	(n.u.)
`dsName`	8	Data stream name
`fill`	8	(n.u.)

The data stream queue header, based at `dsAddr`, has a typical 32-byte structure as follows:

Field	*Size*	*Meaning*
`dsType`	2	Header type, always 1.
`eSize`	2	Record size.
`uOff`	2	User header offset.
`dsSize`	2	Total count of records written.

`tCount`	4	Total count of records ever written into data stream.
`spare`	4	(n.u.)
`IN`	2	Offset (from start of queue header) to next record.
`LIMIT`	2	Offset to end (= *DS_size*).
`START`	2	Offset to first record (following header).
`extra`	2	(n.u.)
`uHdr`	8	User header space (*Size* = *User_size*).

The total size of the header is allocated from the start of the space defined for the data stream queue. Its size is usually 32 bytes, but it can be made larger if more user header is specified. (The latter may be used for anything of interest for the specific data stream, although it is usually not used at all. A user size of 8 bytes implies a total queue header size of 32 bytes.)

The sign bit of the first word of the `DSTRM` entry is set when the queue has been initialized. It must be set to enable the `DSWrite` routine to write a record into the queue.

The record size may be set to zero, implying that variable size records can be written. None of the data streams described in this note uses variable size records.

The offset where the next record will be written is specified by `IN`. If the size of the record cannot fit in the space remaining in the queue, as given by `LIMIT`, this offset is set to `START` and the record is written there. (A record must be written contiguously.)

Note that this queue design does not provide an `OUT` offset that could hold up writing additional records if the queue reader gets too far behind. The queue may be read by multiple users without "consuming" the records. Each user must remember its own `OUT` offset. Often, a data stream queue is "read" by a data request specifying one of the listypes defined for the purpose, and the `OUT` offset is automatically retained within the request support structure. Each user that wants to monitor records written into the queue must ensure that it reads it often enough to "keep up."

Back to the functionality of the various data streams in common use:

NETFRAME

Each 16-byte record describes a datagram received or transmitted. Its structure is:

Field	*Size*	*Meaning*
`buffer`	4	Address of start of frame buffer (+ 18)
`dSize`	2	Datagram size, actually the total ethernet frame size – 18.
`pNode`	2	Node number, usually a pseudo node number.
`time`	8	Time-of-day in BCD, as `yr-mo-da-hr-mn-sc-cy-ms`.

Although the buffer address is given, the area of memory used for circular frame buffers is quite dynamic, so it is possible to see the given datagram contents only if one does not wait so long that the frame buffer "wraps." In the 68K-based IRMs, there are currently 340 ethernet 1500-byte receive frame buffers that occupy 512K bytes. The number of transmit frame buffers depends upon the size of each frame transmitted, but the total memory set aside for transmit buffers is 96K bytes.

The pseudo node number is a scheme to identify a node in 16 bits by referring to an entry in the `IPARP` table that holds an IP address and an associated UDP port number.

The BCD time-of-day format is commonly used in these front ends to make it readable in hex. In this case, the `yr` byte holds a flag to indicate whether the datagram was transmitted (`0xC0`) or received (`0xC1`). The `cy` byte is the 15 Hz cycle number (range `00–14`). The `ms` byte is a binary number of half milliseconds since the start of the cycle when the datagram was transmitted or received. For a node running in 15 Hz cycles, it should not be more than `0x86`.

The page application `NETF` supports access to this data stream queue in a target node. It also provides "friendly node numbers" where appropriate, replacing the dynamically assigned pseudo node numbers when it knows the native or Acnet node number for the IP address and port number used. It lists the results to a serial port. See note, *Network Frames Page*, and also *Network Frames Addition*.

SETLOG

Setting log records describe recent settings that have been made. In these front ends, the notion of a "setting" is quite general and does not always relate to hardware D/A's. For example, a setting might send a line of characters to a target node serial port, or it might target an arbitrary area of memory. The 16-byte setting log record is as follows:

Field	*Size*	*Meaning*
`sNode`	2	Pseudo node# that sent the setting message
`ltyNby`	2	Listype#, #bytes of setting data
`idDat`	4	ident index/data, or address ident
`time`	8	time-of-day in usual BCD format

If the setting is issued internally by the local node, `sNode` is the local native node number; otherwise, it is a pseudo node number. If such a setting is issued from a local application, the sign bit of the listype# byte is set. If the ident is 4 bytes long, such as in an analog channel number, `idDat` is the last 2 bytes of the ident part followed by the first 2 data bytes. (This may be a channel number followed by the setting data.) If the ident is 6 bytes, the last 4 bytes of the ident are recorded, without any data bytes. This may be for a memory address ident, where the entire address is used. See the note, *Settings Log Implementation*, for more details.

Access to the `SETLOG` data stream is supported via `PAGESLOG`, which can list the results. See the note, *Settings Log Page*, for more details on this page application.

ACNETERR

This data stream holds 16-byte records of `RETDAT` errors as observed by a server node. A server node handles a request on behalf of a requesting node, taking care to gather the requested data from one or more nodes before returning a composite reply message. As a kind of middleman, it can monitor the timely replies from each contributing node to an active request. Most often, the errors recorded reflect missing replies from one or more contributing nodes. If missing replies persist, the server node resends the original request to the missing node, hoping that if it is down, it will soon return to service and provide its portion of the data to a given active request. Such resends are provided by the server node for both periodic and clock event-based requests. A record of each resend is written into the `RESENDDL` data stream, described later. The `ACNETERR` record format is:

Field	*Size*	*Meaning*
`rNode`	2	Requesting Acnet node#
`dSize`	2	Reply message data size
`eStat`	2	Acnet error status word (first nonzero value)
`mNode`	2	Missing friendly node#
`time`	8	BCD time-of-day of the error-laden reply

No page application supports specific access to this data stream, but a local application called `MISS` can be used to monitor this data stream (even from another node) by capturing 2K bytes of "data of interest" as soon as it detects a new record in the server node. The data captured includes the original data stream record, the last 16 Booster reset clock events as seen by the node running `MISS`, the last 32 clock events seen by the missing node, the last four `RETDAT` requests seen by the missing node, the last 24 `NETFRAME` records logged by the missing node, and the last 96 `NETFRAME` records logged by the server node. This has been used extensively to analyze Acnet errors detected by Booster HLRF server `node06CF`. Listing out the captured records was done via `PAGEPMEM`. See the note, *Missing BRF Replies*, for more on `MISS`. See the note, *Acnet Error Log*, for more on `ACNETERR` itself.

TFTPLOG

This data stream logs `TFTP` protocol transactions as supported by the `TFTP` protocol server local application called `TFTP` (actually, `LOOPTFTP`), which can support up to 10 simultaneous `TFTP` transactions. In practice, this is most useful in `node0562`, from which the IRM front ends boot. In that case, the file name that is requested is called `SYSTEM`, and `node0562` knows where to find that code to return. Currently, the transfer of the 210 records takes about 1.2 seconds. (That node, following a power interruption, gets the system code from its own flash memory.) Another node that is commonly targeted via the `TFTP` protocol is library `node0508`. When MPW is used to build a new local or page application, it is usually downloaded to `node0508` via the `TFTP` protocol. The reception of such a program file causes assignment of the version date, which is why it is normally only sent to the library node via `TFTP`. From the library node, `PAGEDNLD` is used to copy it via Classic protocol (preserving the version date) to other nodes. The 16-byte records are as follows:

Field	*Size*	*Meaning*
`ipAddr`	4	IP address of node initiating `TFTP` transfer.
`nRec`	2	#records in transaction.
`eTime`	2	elapsed time of transaction, in ms.
`time`	8	time-of-day in usual BCD format.

The note describing this is *TFTP Server Log*.

Access to these records is supported via the page application `TFTP`. See the note, *TFTP Server Log Page*, for details.

EVTLOG

Every clock event causes an interrupt, and each is logged into a data stream. A 4-byte record consists of a clock event number byte followed by a 3-byte time stamp in μs units.

The data stream is viewed by `PAGEEVTQ`, which can produce a detailed listing of clock events and associated timing. See the note, *Clock Event Diagnostics*, for more details on this. Other related notes are *Clock Event Handling* and *Clock Events Page* (for `PAGEEVTS`).

TASKLOG

Each 16-byte record in this data stream marks the execution of a system task. The records have the following format:

Field	*Size*	*Meaning*
`tMask`	2	Mask indicating the task; *e.g.*, `0x0040` is task 6, the `Update` task.
`events`	2	Task event bits awaiting execution.
`eTime`	4	Elapsed execution time, in μs.
`time`	8	time-of-day in usual BCD format.

A task-switch callout procedure is used to collect this data for logging into the data stream. See the note, *Task Timing for IRMs*, for more. PowerPC nodes use a similar approach.

Page application `PAGETASK` supports access to these records for displaying/listing. See the notes, *Task Activity Analysis* and *Task Activity Example*, for more on this, as characterized for its use for 68K-based IRM front ends. For PowerPC-based systems, use the PowerPC version of `PAGETASK` to monitor task activities in targeted PowerPC-based nodes.

AERSLOG

This data stream logs records of communications with Acnet alarm handler `AEOLUS`. This can help resolve questions about Acnet alarm reporting, where each front end sends its alarm messages to `AEOLUS`, but not too fast. The necessary shepherding logic is handled by local application `AERS`. The 16-byte record format is:

Field	*Size*	*Meaning*
`rType`	2	Record type#.
`messId`	2	Message Id used for communications
`info`	4	Data depending on type#.
`time`	8	Usual BCD time-of-day.

Here are the `info` fields that depend upon the record type#:

Type#	*Meaning*	*Data*
0	`FEBT` message sent	0
1	`FEBT` ack	(Status, Ack time cycles)
2	`ERM` message sent	(retry#, #packets)
3	`ERM` ack	(Status, Ack time cycles)
4	Big Clear received	0

When `AERS` is initialized, or following detection of a lack of response from `AEOLUS`, it seeks to determine when `AEOLUS` is alive again. This is done by sending, periodically, a `FEBT` message. Upon receiving an acknowledgment reply, it then enters a state to await alarm messages passed from the underlying system `Alarm` task (via a message queue) to `AERS`, queuing them for delivery to `AEOLUS`, packaging up to 22 of them in a single `ERM` message. If it fails to get an acknowledgment from the `ERM` message, it retries at least 2 more times, with plenty of time between, before giving up and reverting back to the state of sending `FEBT` messages until it receives an acknowledgment. After receiving an `ERM` ack, it "moves on." The various delays that characterize its timely behavior come from the `AERS` parameters. Relevant notes are

Acnet Alarms via DI and *SRP Support for Acnet*.

No page application supports these `AERSLOG` records. One can examine them via the Memory Dump page, `PAGEMDMP`, or one can get a listing via the Print Memory page, `PAGEPMEM`.

RETDLOG

This data stream holds a record for each `RETDAT` data request received by the `ACReq` task. Each 16-byte record has the following format:

Field	*Size*	*Meaning*
`rNode`	2	Acnet node number of requesting node.
`dSize`	2	Reply data size, including status word(s).
`nDev`	2	Number of device-properties in the request.
`ftd`	2	Frequency-time descriptor.
`time`	8	Time-of-day message processed.

Note that three fields (`dSize`, `nDev`, `ftd`) are taken from the `RETDAT` request message header.

For the case of a cancel message, these three fields are set to zero.

The time-of-day `yr-mo` bytes are overwritten with the request id taken from the Acnet header. This helps to correlate the cancel message with a previous request message.

For this data stream the 8-byte user header area is useful for some tests. The 4 words allow writing (probably via the Memory Dump page) one or more Acnet node numbers to define a filtered selection of nodes for which records are written. There are two options: If the first of the 4 words is nonzero, it means up to 4 nodes are given whose requests *should* be recorded. If the first word is zero, it means that up to 3 nodes are given whose requests *should not* be recorded. If all words are zero, no filtering is used, so that all requests are logged. This was done especially for filtering out routine requests from a `DAE` that merely serve to monitor the continued health of the front end, including its ability to respond to `RETDAT` requests.

No page application directly supports the `RETDLOG` data stream. For much more detail than one probably wants to know, see the 14-page note, *RETDAT-SETDAT Support*.

IPNODLOG

This data stream relates to the network lookup logic that is designed to avoid searching nonvolatile memory tables, which is especially slow in PowerPC nodes. Instead of searches, table lookups are used. This one is somewhat complicated, so bear with me. There are other notes available. From a a paragraph at the end of the note, *Network Table Lookup*:

> A document describing the evolution of the scheme is called *Table Lookup Improvement*. A document describing the communications between local applications and the system code *vis-à-vis* changes needed for `IPNOD` table entries is called *Table Lookup IP Address*. A document detailing the changes required to the system code to support the table lookup scheme, plus preliminary timing results, is called *Network Table Changes*.

Records in this data stream are of three different types: announcement, advisory, and diagnostic. Here is the record format:

Field	*Size*	*Meaning*
`ipAddr`	4	IP address.
`newV`	2	New node number value.
`oldV`	2	Old node number value.
time	8	time-of-day.

An announcement record is written by the `AAUX` or `DNSQ` local application when it initializes or terminates. The system code, via the routine `MONIPNOD`, in module `IPNODGS`, watches for such records and sets or clears flag bits in the low memory global variable called `IPNODDSX`. The idea is to keep track of whether these LAs are active. The announcement record is identified by having both the `ipAddr` and `newV` fields set to zero. In addition, if the low byte of `oldV` is either `0x4E` (‘`N`’ for `DNSQ`) or `0x41` (‘`A`’ for `AAUX`), it is an announcement record. If the high byte is `0xFF`, it is an “open for business” announcement; else, it is a termination announcement.

If it is not an announcement, an advisory record is recognized by the `oldV` value being `0xFF4E` (from `DNSQ`) or `0xFF41` (from `AAUX`). Such a message from `DNSQ` indicates a new native node number association with an IP address in the `ipAddr` field. If it is from `AAUX`, it implies a new Acnet node number association with an IP address. These announcements serve to keep the `IPNOD` network lookup system table up-to-date.

A diagnostic record, recognized as one that is neither an announcement nor an advisory record, is used to record a change in a node number associated with an IP address. The change must be from an old nonzero value to a new (different) nonzero value.

Another unusual characteristic of this data stream is that it actually makes use of the user area in the header, which has been set to 40 bytes. In that area is room for four 8-byte records that contain timing information about the last network table lookup operation, separated into the four types of lookup routines that do this. This reflects the fact that the whole purpose of developing the required data structures to support the table lookups was to make them more efficient, by reducing the number of nonvolatile memory accesses required. This timing information thus shows “the proof of the pudding.” The four routines for which timing is monitored are, in order, as follows:

`PSNIPARP`	Given an IP address and UDP port#, return a pseudo node#.
`IPNODEN`	Given a native node#, lookup its IP address, and return pseudo node#.
`GTNODEN`	Given an IP address, return native node#.
`FINDUDP`	Given a pseudo node#, lookup its IP address, return Acnet node#.

For each case, the format of the 8-byte timing record is as follows:

Field	*Size*	*Meaning*
`nodeNum`	2	Node# logged.
`countr`	2	Counter of #times record updated.
`mxTime`	2	Maximum elapsed time, in μs.
`lsTime`	2	Most recent elapsed time.

No page application supports this data stream. Consult the many notes listed above.

`ITTLOG`

The `ITTLOG` data stream supports logging of interrupt timing diagnostic records. Each kind of interrupt results in a diagnostic record of timing information being written into the `ITT` (Interrupt Timing Table). Each such entry has some flag bits, one of which enables the logging of the information in that record every time it is updated, thus providing real time information about multiple interrupt occurrences. For more detail on this, see the two notes, *Interrupt Timing Diagnostic* and *ITT Operations*.

ALLOCLOG

This data stream is used in PowerPC nodes to log each memory block allocation and release. All system and application calls to allocate or free memory use the `AllocP()` and `FreeP()` "wrapper" functions to do this, writing a record to this data stream before/after calling `malloc()` or `free()`. The format of the 16-byte record is as follows:

Field	*Size*	*Meaning*
`blkAddr`	4	Allocated block base address
`blkSize`	2	Allocated block size
`flags`	1	Flags
`blkType`	1	Allocated block type#
`time`	8	Usual BCD time-of-day

The `flags` byte field indicates in its low nibble certain potential consistency check failures that might occur during a call to `FreeP()`. Its high nibble indicates `0xF` if the call was to free memory, or `0xA` if the call was to allocate memory. See the note, *Memory Allocation Diag*, for more details on this.

The page application `MEMB` monitors this data stream in a target (PowerPC) node, and it updates a little console display showing the current blocks that have been allocated but not yet freed, from the time that the page application monitoring was started. It maintains this list in order of age, with the oldest first. It also shows the type of each block. In addition, if `MEMB` is used to monitor the local node, it can also show the requesting node# for recognizable request block types. One really has to see this to appreciate it. See the note, *Memory Allocation Page*, for more details on `PAGEMEMB`.

BUSERROR

In the PowerPC systems, this data stream is used to log bus error occurrences as reported via `vxMemProbe()` calls. All system and application code calls `MemProbe()` as a wrapper around the call to `vxMemProbe()` in order to accomplish this. Such logic is designed to prevent a bus error occurrence from disabling a task and thus the front end operation. A note, *PowerPC Bus Errors*, discusses this further. Here is the 16-byte record format:

Field	*Size*	*Meaning*
`taskNum`	1	current task number in range 0–15
`rwLength`	1	flag `0x10` if write; length 1,2,4 in ls 3 bits
`eTime`	2	elapsed time for the call to `vxMemProbe()`, in TB units
`addrs`	4	address whose access produced bus error
`time`	8	time of occurrence in usual BCD format plus `ms`

There is no page application designed to display the `BUSERROR` records. One expects that this data stream should not see frequent activity. But that's the reason for the diagnostic.

FTPMLOG

This data stream records the FTPMAN protocol requests processed by a node. It is somewhat analogous to RETDLOG that is used for RETDAT protocol requests. The format of its records are as follows:

Field	*Size*	*Meaning*
rqNode	2	Requesting node#
rqType	1	FTPMAN protocol type code
rqNDev	1	#devices in request
rqNPts	2	#pts/device if snapshot, #pts/reply/device if continuous
rqDTime	2	elapsed time to process request, in μs
rqMsgId	2	message-id specified by client
rqTime	6	time-of-day in BCD, as Da-Hr-Mn-Sc-Cy-ms

For more details, see the note, *FTPM Log*.

RESENDDL

This data stream is used in data server nodes for records of requests that have to be resent to a contributing node from which replies are consistently missing. Its 16-byte records have the following format:

Field	*Size*	*Meaning*
rsndNode	2	target node for resent request
rsndSize	2	size of data requested
rsndDev	2	#devices in request
rsndFTD	2	frequency time descriptor
rsndTime	8	time-of-day in usual BCD format

For a bit more on this, see note, *Acnet Resend Log*.

SETDLOG

This data stream holds records about recent SETDAT protocol messages processed by the local node. The 16-byte records have the following format:

Field	*Size*	*Meaning*
sdNode	2	Source node# for SETDAT message
sdMsgId	2	Source message id from Acnet header
sdNDev	2	#device-properties in SETDAT message
sdExTime	2	Execution time for setting action, in μs
sdTime	8	Usual BCD time-of-day as yr-mo-da-hr-mn-sc-cy-ms

The yr byte of the time-of-day is used for holding some flag bits. More details on this diagnostic are found in the note, *SETDAT Diagnostic*.

Request support for data streams

Access to data stream records is supported by several listypes, as follows:

Listype	*Record access type*
50	Future records
51	Most recent records
52	Queue header
53	`DSTRM` table entry
54	Name
55	Generic name lookup
78	Oldest records

Listype 50 returns any new records written into the data stream as of the time the request is received. This is normally a periodic request. The requester of data stream records always has the responsibility of "keeping up," so that no records will be missed. This means choosing the reply period to be short enough and the number of bytes requested to be large enough. Be aware that the length requested should allow for a 4-byte header that consists of a count of the number of records included in the buffer following the header and the size of each record. It may often be that the count is zero, which merely means that no new records have been written since the previous reply.

Listype 51 returns the most recent records that can fit within the number of bytes in the request. This is most often used as a one-shot request.

Listype 52 provides access to the data stream queue header.

Listype 53 provides access to a `DSTRM` table entry.

Listype 54 provides access to the name field of a `DSTRM` table entry.

Listype 55 allows finding out which data stream record matches a given name. To use this generic name lookup listype for data streams, use the type number 3 in the second word of the 12-byte ident, following the node number (zero for a multicast request) and preceding the 8-byte (blank-filled) name. The reply gives the local node number and the data stream index if there is a match, else it returns zeros for both words.

Listype 78 is similar to listype 51, but it begins with the oldest record in the data stream queue circular buffer. It thus allows access to all records in the queue that are available.